# A New Parallel Version of the DDSCAT Code for Electromagnetic Scattering from Big Targets

**R. W. Numrich**[1], **T. L. Clune**[2], **and K.-S. Kuo**[2, 3]

[1]The Graduate Center, City University of New York, New York, New York, USA
[2]NASA Goddard Space Flight Center, Greenbelt, Maryland, USA
[3]Earth System Science Interdisciplinary Center, University of Maryland
College Park, Maryland, USA

**Abstract**— We have modified the publicly available electromagnetic scattering code called DDSCAT to handle bigger targets with shorter execution times than the original code. A big target is one whose sphere equivalent radius is large compared to the incident wavelength. DDSCAT uses the discrete-dipole approximation and an accurate result requires the spacing between dipoles to be small relative to the wavelength. This requirement, along with the big target size, implies that the number of discrete grid cells defining the computational grid box surrounding the target must be large. The memory requirement is thus significant and the execution time long. The original version of the code cannot handle large enough targets that we would like to consider because they require memory exceeding what is available on a single node of our cluster and execution time extending to days.

We have removed two limitations of the original version of the code. First, to speed up the execution involing multiple target orientations, the original code assigned one, but only one, MPI process to each independent orientation. We surmount this limitation by assigning each orientation a team of processes. Second, the original code allocated all the memory for the full problem for each MPI process. We surmount this limitation by decomposing the data structures across the members of a team. With these changes, the new version can handle much bigger targets. Moreover, it exhibits strong scaling for fixed problem size. The execution time decreases very nearly $1/p$ as the processor count $p$ increases. Execution time measured in days with the original code can now be reduced to fractions of an hour.

## 1. INTRODUCTION

The discrete-dipole approximation (DDA), based on the original work of Purcell and Pennypacker [7], is a standard method for calculating electromagnetic scattering cross sections of irregularly shaped targets [6]. One of the more effective codes for this kind of calculation is the DDSCAT code from Draine and Flatau [1]. A comparable one is the ADDA code from Yurkin and Hoekstra [9]. Kahnert provides a review of other solution methods in a recent review paper [3].

DDA reduces the scattering problem to the solution of a large, dense, complex symmetric system of linear equations,

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

Design of a code to solve these equations and the resulting performance of the code depend on the data structures chosen to represent the matrix and the solution technique selected to solve the system of equations. Since the matrix is a symmetric Toeplitz matrix, only the elements of a single row need be stored, and these elements can be decomposed and distributed across processors in a straightforward way. To solve the system of equations, the community has, after many years of expermentation and analysis, settled on using one or another variation of Krylovbased solvers [8, Chs. 6–7]. These iterative solvers require several matrix-vector multiplications at each iteration. Since the matrix is Toeplitz, representing a convolution, this operation can be performed using Fast Fourier Transforms. The matrix-vector multiplication, then, becomes a forward FFT of two vectors followed by multiplication in Fourier space followed by inverse FFT back to physical space.

The parallel strategy adopted in the original version of the code assigns a separate MPI process to each independent target orientation. Each MPI process allocates all the memory for the full problem, which may exceed the memory available on a node for big targets, and the number of processes that could be used is limited by the number of orientations. The new code assigns a team of MPI processes to each independent orientation and partitions the data structures by the size of the team. The problem now fits in memory and more processes can be used to reduce the execution time.

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE | 2. REPORT TYPE | 3. DATES COVERED |
|---|---|---|
| **MAR 2013** | **N/A** | **-** |

4. TITLE AND SUBTITLE
**A New Parallel Version of the DDSCAT Code for Electromagnetic Scattering from Big Targets**

5a. CONTRACT NUMBER
5b. GRANT NUMBER
5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER
5e. TASK NUMBER
5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
**The Graduate Center, City University of New York, New York, New York, USA**

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)
11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release, distribution unlimited**

13. SUPPLEMENTARY NOTES
**See also ADA584538. Progress in Electromagnetics Research Symposium (33rd) (PIERS 2013) held in Taipei, Taiwan on 25-28 March 2013. U.S. Government or Federal Purpose Rights License**

14. ABSTRACT
**We have modi¯ed the publicly available electromagnetic scattering code called DDSCAT to handle bigger targets with shorter execution times than the original code. A big tar- get is one whose sphere equivalent radius is large compared to the incident wavelength. DDSCAT uses the discrete-dipole approximation and an accurate result requires the spacing between dipoles to be small relative to the wavelength. This requirement, along with the big target size, implies that the number of discrete grid cells de¯ning the computational grid box surrounding the target must be large. The memory requirement is thus signi¯cant and the execution time long. The original version of the code cannot handle large enough targets that we would like to consider because they require memory exceeding what is available on a single node of our cluster and execution time extending to days. We have removed two limitations of the original version of the code. First, to speed up the execution involing multiple target orientations, the original code assigned one, but only one, MPI process to each independent orientation. We surmount this limitation by assigning each orientation a team of processes. Second, the original code allocated all the memory for the full problem for each MPI process. We surmount this limitation by decomposing the data structures across the members of a team. With these changes, the new version can handle much bigger targets. Moreover, it exhibits strong scaling for ¯xed problem size. The execution time decreases very nearly 1=p as the processor count p increases. Execution time measured in days with the original code can now be reduced to fractions of an hour.**

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT<br>**unclassified** | b. ABSTRACT<br>**unclassified** | c. THIS PAGE<br>**unclassified** | **SAR** | **5** | |

**Standard Form 298 (Rev. 8-98)**
Prescribed by ANSI Std Z39-18

## 2. THE DISCRETE-DIPOLE APPROXIMATION

DDA surrounds the target with a three-dimensional computational grid box discretized into cubical cells of volume $d^3$ with a dipole located at the center of each of the cells occupied by the target [7]. If there are $n$ dipoles in the target, the volume occupied by these dipoles is $nd^3$, resulting in an sphere-equivalent radius of $a$, defined by the relationship, $nd^3 = (4\pi/3)a^3$. Scattering cross sections are often measured in units equal to the cross sectional area, $\pi a^2$ [2, Ch. 16, 4, Ch. 2].

The accuracy of the method requires [1, 10, 11] that the grid spacing between cells, $d$, be small relative to the wavelength,

$$2|m|kd < 1 \tag{2}$$

where $m$ is the complex refractive index of the target and $k = 2\pi/\lambda$ is the wavenumber.

A target is considered "big" if its *size parameter*, $ka$, is much bigger than one, i.e., $ka \gg 1$. Coupled with requirement (2), it implies a large number of dipoles, $n > (2\,|m|\,ka)^3$, and almost always an even larger number of grid cells. The memory requirement for a large target may thus exceed what is available on a single computer or a node in a cluster. Furthermore, the computational work that must be done to solve the system of equations increases with the size of the grid box and the execution time for large targets may easily expand to days if only a single node can be used. To address these two problems, we decompose the computational box such that each partition of the box fits into the memory of a single node, and we assign a team of processors to work in parallel on each partition to reduce the execution time.

## 3. THE PARTITIONED COMPUTATIONAL PROBLEM

The parallel implementation strategy for any application code involves two issues: how to partition data structures and how to assign work. The original code takes into consideration one of these issues, the assignment of work, but does not consider the partition of data structures. The new version addresses both issues.

The details of the partition depend on the details of how the system of Equation (1) is represented on the grid points of the discretized computational grid box. The vector on the right side is proportional to the incoming plane wave, $\exp(i\mathbf{k}\cdot\mathbf{r})$, with wavelength $\lambda$, wavenumber $k = |\mathbf{k}|$, and the direction of the wave vector relative to the target by three Euler angles $(\alpha, \beta, \gamma)$. At each grid point of the computational grid box, the vector on the right side of the system of equations, $b_{ijk} = \exp[i\mathbf{k}\cdot(r_i, r_j, r_k)]$, has values proportional to the value of the plane wave at each grid point $\mathbf{r} = (r_i, r_j, r_k)$. In the same way, the values of the elements of the matrix $\mathbf{A}$ are evaluated relative to the grid points, and the solution vector $\mathbf{x}$ is found at each grid point. In practice, indices for the points on the three-dimensional grid are serialized so that the quantities $\mathbf{b}$ and $\mathbf{x}$ can be thought of as vectors of length $n^3$ and the matrix $\mathbf{A}$ can be thought of as a matrix of size $n^3 \times n^3$.

The decomposition strategy is straightforward. The vectors involved are partitioned along one of the directions of the computational grid box. If the grid points have been serialized in lexical order, the $x$-direction first, followed by the $y$-direction and then by the $z$-direction, the vectors can be partitioned in the $z$-direction by the team size (i.e., number of processes per team) assigned to the partition. Correspondingly, the rows of the matrix are partitioned along the $z$-direction. This partitioning introduces overhead into the code due to the requirement for moving distributed data structure among team members, but for big targets, this overhead is small as our results show in later sections.

## 4. PARALLEL CONVOLUTION

Special properties of the matrix $\mathbf{A}$ make the problem tractable. The matrix is a function of the magnitude of the plane wave vector, $k = |\mathbf{k}|$, not its direction, i.e., $\mathbf{A} = \mathbf{A}(k)$. The matrix can thus be calculated once for each wavelength and used for all orientations of the incoming wave. It is also a Toeplitz matrix. The values of its elements depend on the distance between grid points such that, in the serialization representation, the matrix elements have the property, $a_{ij} = a_{i-j}$. It can then be represented by one of its rows, a vector of length $n^3$, rather than a matrix of size $n^3 \times n^3$.

The vector $\mathbf{b}(\mathbf{k})$ and the solution vector $\mathbf{x}(\mathbf{k})$, on the other hand, are necessarily functions of both the magnitude and the orientation of the incoming wave. Thus, for each orientation, the matrix remains the same but the right-hand-side vector changes so the system of equations,

$$\mathbf{A}(k)\,\mathbf{x}(\mathbf{k}) = \mathbf{b}(\mathbf{k}), \tag{3}$$

must be solved for the solution vector for each oreientation.

When iterative Krylov solvers are used to solve this system, several matrix-vector multiplications of the kind: $\mathbf{y} = \mathbf{A}\mathbf{x}$, must be performed for each iteration. Since the matrix is Toeplitz, its elements depend only on the difference of the indices, and the matrix-vector multiplication becomes a convolution, $y_i = \sum_j a_{i-j} x_j$. The operation count for this multiplication is on the order of the square of the rank of the matrix if done in the usual way. But since the Fourier transform of a convolution is the product of the Fourier transforms of the the two factors, $\mathcal{F}(\mathbf{y}) = \mathcal{F}(\mathbf{A}) \cdot \mathcal{F}(\mathbf{x})$, the operation count can be reduced to the order of the matrix size times the logarithm of the matrix size. This difference results in a signficant saving of computation time. The result vector is retrieved with the inverse transform, $\mathbf{y} = \mathcal{F}^{-1}[\mathcal{F}(\mathbf{A}) \cdot \mathcal{F}(\mathbf{x})]$.

The Fourier transform is a three-dimensional transform over the grid points of the computational box. Since the box has been decomposed along the $z$-direction, data communication among processors is required, which is not required on a single processor. Each processor can perform the transform independently for the grid points it owns in the $xy$-plane, but the data structure must be transposed to do the transform in the $z$-direction. The matrix can be transformed once for each wavelength and left in the transposed data structure in Fourier space. It does not need to be transformed back to the original decomposed data structure. The vectors at each iteration of the Krylov solver are also left in transformed Fourier space and multiplied by the matrix. Only then is the product of the two vectors transformed back to the original partitioned data structure.

An important optimization of the convolution step results from treating the zeros that must be padded onto the vectors before performing the Fourier transform [5]. Total time for the transform, using a Fast Fourier Transform (FFT) library, is the sum of the time in each of the three dimensions, $t_1 = t_x + t_y + t_z$. If the dimensions are about the same in each dimension, the time for the transform is the same in each direction, $t_x = t_y = t_z = t_0$, and $t_1 = 3t_0$. During the FFT in the $x$-direction, the time reduces by half by not computing over the zeros in the $y$-direction and again by half by not computing over the zeros in the $z$-direction. Likewise, in the $y$-direction, the time reduces by half by not computing over the zeros in the $z$-direction. The time in the $z$-direction remains the same so the reduced time becomes, $t_2 = t_0/4 + t_0/2 + t_0 = (7/4)\, t_0$, and the speedup in performance becomes $t_1/t_2 = 3t_0/(7t_0/4) = 12/7 \simeq 1.71$.

For large targets, where the FFT dominates the execution time, this modification of the code may lead to signicant performance speedup. For smaller targets, where the FFT is only a fraction of the total execution time, the benefit will be lower. If the transpose represents 50% of the execution time, the benefit may only be about 20%. We have measured a 12% benefit per grid point on a single processor for a test case.

## 5. PARALLEL MATRIX TRANSPOSE

For multiple processors, the analysis of the time saved due to the treatment of the padding is somewhat different from that of a single processor due to the nontrivial cost for the matrix transpose. The amount of data transferred between the $xy$-direction to the $z$-direction, during the matrix transpose reduces by a factor of 2. For large problem sizes, we have observed that the time required for the transpose was comparable to the total time for the FFT. If the transpose is bandwidth limited, the net speedup for FFT with transpose can be estimated as $t_1/t_2 = (3t_0 + 3t_0)(3t_0/2 + 7t_0/4) = 24/13 \simeq 1.85$. We have, in fact, measured an overall speedup equal to 1.7× with the new treatment of the padding.

## 6. SCALING ANALYSIS

To compare performance of the old version of the code with the new version, we examine two different targets referred to here as *medium* and *large*. The medium target fits within the memory of a single node on our machine, and allows direct performance comparisons between the two code versions. The large target, does not fit in the memory of a single node, and can only be run with the new version. The large target illustrates that the new version of the code can handle larger targets and that, for these big targets, it exhibits strong scaling.

An important advantage of the new implementation, is that it allows the code to utilize far more processors. Whereas the original code limited the number of MPI processes to the number of target orientations, typically $O(10^2)$ the new version uses a team of processes for each orientation. The number of MPI processes in a team is only limited by the size of the computational grid.

Parallel strategy for the new version of the code is good for the large case which does not fit in the memory of a single node. With the new version of the code, we use teams of size 20 and distribute the MPI processes across nodes with each process requiring one-twentieth of the memory.

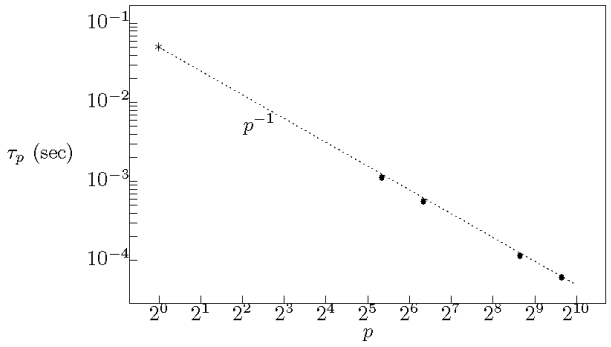


Figure 1: Execution time per grid point, $\tau_p = t(p)/n_x n_y n_z$, as a function of processor count. Large target with $|m|kd = 0.3563$ contained in a box of size $(n_x, n_y, n_z) = (256, 360, 360)$ marked with bullets $(\cdot)$ measured in seconds. The dotted line represents perfect scaling of $p^{-1}$. The intercept at $\log(\tau_1) = -1.3$, marked with an asterisk (*), suggests that the problem on a single processor would require at least 19 days to complete compared with about half an hour on 800 processors.

For this problem, the FFT dominates the execution time taking almost 99% of the total. This part of the code is fully parallelized and the execution time, for the fixed problem size, exhibits strong scaling — decreasing inversely with the processor count as illustrated in Figure 1.

## 7. ACCURACY OF THE METHOD

The accuracy, as well as the performance, of codes that implement the discrete-dipole approximation depends on a number of empirically determined parameters [10, 11]. The distance between cells, for example, must be small relative to the wavelength of the incident wave as displayed in inequality (2). The rate of convergence of the iterative solver also depends on the value of this parameter, smaller values generating larger matrices requiring more iterations to converge.

Now that the execution time has been reduced to a few minutes of time for very big targets, we can perform parameter studies to test the accuracy of the results. In particular, we can follow the technique suggested by Yurkin and coworkers [11] to extrapolate the results to zero cell size.

## 8. SUMMARY

The major changes to the code, after decomposing the data structures, required a 3D transpose for the FFT and a global reduction for the scalar products and norms in the iterative solver. Data decomposition was by far the hardest change to make to the original code. Performance of the code improved when we removed unnecessary calls to the FFT routine for padded zeros required for the convolution. For big targets, the new code exhibits almost perfect strong scaling as the processor count increases.

## ACKNOWLEDGMENT

The authors are grateful to the NASA High-End Computing program for making this improvement possible.

## REFERENCES

1. Draine, B. T. and P. J. Flatau, "Discrete-dipole approximation for scattering calculations," *J. Opt. Soc. Am. A*, Vol. 11, No. 4, 1491–1499, 1994.
2. Jackson, J. D., *Classical Electrodynamics*, 1st Edition, John Wyley & Sons, 1962.
3. Kahnert, F. M., "Numerical methods in electromagnetic scattering theory," *J. Quant. Spectrosc. Radiat. Transfer*, Vols. 79–80, 775–824, 2003.
4. Newton, R. G., *Scattering Theory of Waves and Particles*, 1st Edition, McGraw-Hill Book Company, New York, 1966.
5. Nukada, A., Y. Hourai, A. Hishida, and Y. Akiyama, "High performance 3D convolution for protein docking on IBM blue gene," *Parallel and Distributed Processing and Applications: 5th International Symposium, ISPA 2007*, Vol. 4742 of *Lecture Notes in Computer Science*, 958–969, Springer, 2007.

6. Penttilä, A., E. Zubko, K. Lumme, K. Muinonen, M. A. Yurkin, B. Draine, J. Rahola, A. G. Hoekstra, and Y. Shkuratov, "Comparison between discrete dipole implementations and exact techniques," *J. Quant. Spectrosc. Radiat. Transfer*, Vol. 106, 417–436, 2007.
7. Purcell, E. M. and C. R. Pennypacker, "Scattering and absorption of light by nonshperical dielectric grains," *Astrophys. J.*, Vol. 186, 705–714, 1973.
8. Saad, Y., *Iterative Methods for Sparse Linear Systems*, 2nd Edition, SIAM, 2003.
9. Yurkin, M. A. and A. G. Hoekstra, "The discrete-dipole-approximation code ADDA: Capabilities and known limitations," *J. Quant. Spectrosc. Radiat. Transfer*, Vol. 112, 2234–2247, 2011.
10. Yurkin, M. A., V. P. Maltsev, and A. G. Hoekstra, "Convergence of the discrete dipole approximation. I. Theoretical analysis," *J. Opt. Soc. Am. A*, Vol. 23, No. 10, 2578–2591, 2006.
11. Yurkin, M. A., V. P. Maltsev, and A. G. Hoekstra, "Convergence of the discrete dipole approximation. II. An extrapolation technique to increase the accuracy," *J. Opt. Soc. Am. A*, Vol. 23, No. 10, 2592–2601, 2006.